Vexless: A Serverless Vector Data Management System Using Cloud Functions (SIGMOD 2024)

Yongye Su, Yinqi Sun, Minjia Zhang, Jianguo Wang

Presented by Patrick Lee for CSC 2233

February 5, 2024

Designing data systems

- Focus so far on the internal mechanisms of how to make vector databases work
- Zoom out and take a look at the system design for deploying a vector database
- Our options:
    a. Manage our own virtual machines
    b. Have someone else manage the infrastructure
        - This is where cloud functions come in!

AzureVM

Azure Functions

Cloud functions

- FaaS (Functions as a Service)
- Submit code as a payload function
- All operational concerns hidden
- Charged only for function uptime + fixed function invocation overhead cost

Azure Functions

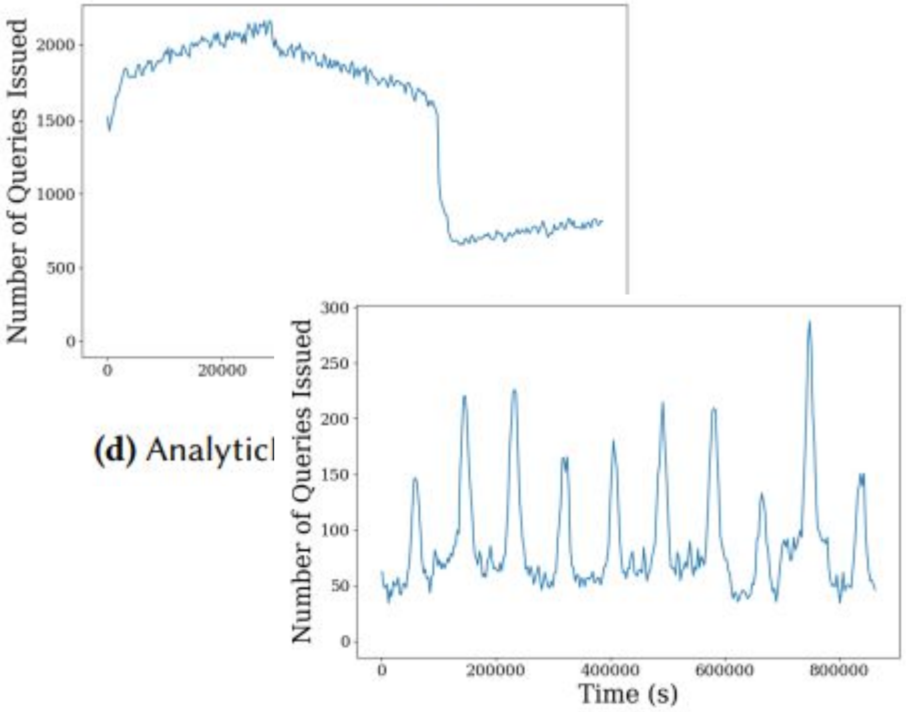
AWS Lambda

Google Cloud Functions

Virtual machines vs. Cloud functions

Functionality	**Virtual Machine**	**Cloud function**
Managed?	Yes	No (serverless)
Dedicated Server?	Yes	No
Stateful?	Yes	No(-ish)
Computational Limits?	Soft (based on machine)	Hard
Start-up time	Minutes	Seconds
Payment model	Rental	Pay-as-you-go
Price-per-unit (of compute)	Lower	Higher

Problem



(d) Analyticl

(e) Twitter Workload [75]

- ANN search
  - Find top-*k* results within *k* results returned by algorithm
- Cloud functions are more expensive than VMs
  - Unsuitable for dense and continuous workloads with long uptime
- Real-world applications of vector databases are sparse and bursty

RQ: Can we use cloud functions to build high-performance and cost-efficient vector databases?

Vexless: 3 Challenges and 3 Contributions

Challenge 1: Hard resource limits in cloud functions, more than one needed

Contribution: Even workload distribution via bespoke sharding algorithm

Challenge 2: Communication (latency) overheads

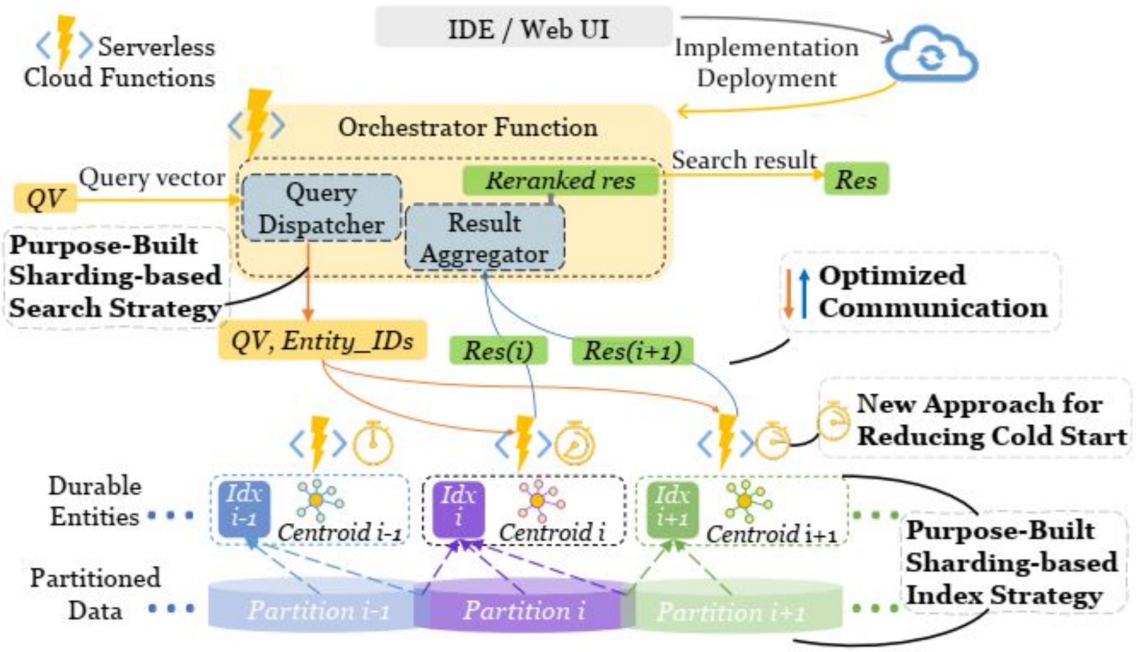
Contribution: Communication hub mechanism

~~Challenge 3: Cold starts~~

~~Contribution: Adaptive scheduling algorithm~~

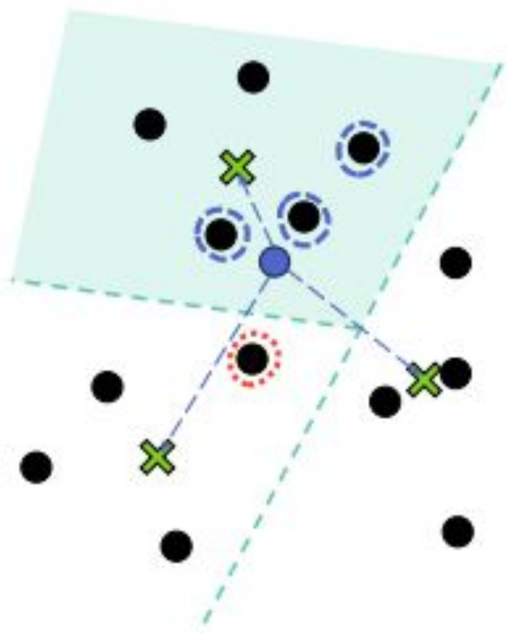
Main System Idea

- Use a distributed vector search, treating each cloud function as a computational unit
  - Sharding phase (partitioning)
  - Search Phase (Identify → Activate → Aggregate)



Naive Sharding Approaches

- Azure has a memory limit of 1.5 GB per cloud function
- Azure functions OOM at 3 million vectors
- How can we split up the database?
  - Uniform sharding
  - Cluster-based (k-means) sharding
  - Balanced k-means sharding



From Moshe Gabel CSC2233 Week 1 Week 1 Slides

Improved Sharding-based Index & Search Strategy

- Balanced K-means problem: Worse search efficiency
- Reason: Forcing boundary points to achieve balance
- Searching in incorrect clusters as a result
- Solution: **radius-threshold-based clustering**

Stage 2: Index Building

Initialization: For cluster c_i, initialize an empty index partition I_i. Set distance threshold T for redundancy indexing.
for each base vector v in D **do**
    Compute v's distances $\mathrm{d}(v, C_i)$ to centroids $C_1, C_2, \ldots, C_k$.
    Rank $\mathrm{d}(v, C_i)$.
    Include v in the index I_h corresponding to its centroid C_h.
    for each centroid C_i where $i \neq h$ **do**
        if $\mathrm{d}(v, C_i) < T$ **then**
            Add v to I_i.
        end if
    end for
end for

Improved Sharding-based Index & Search Strategy

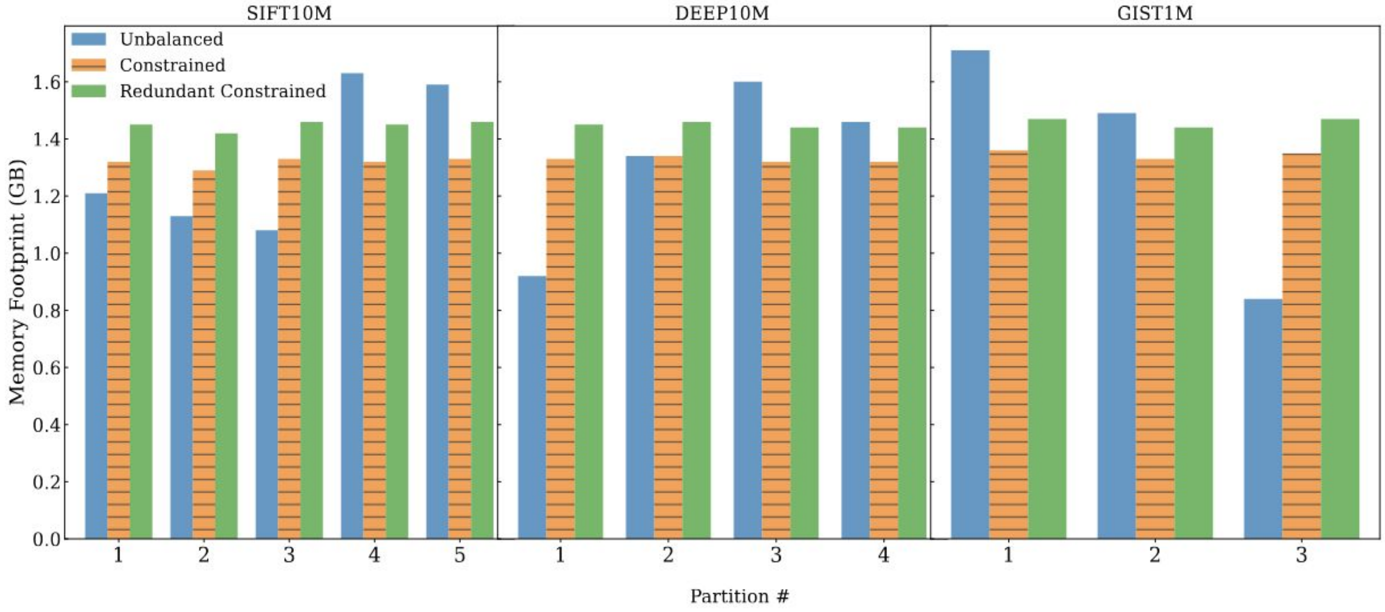
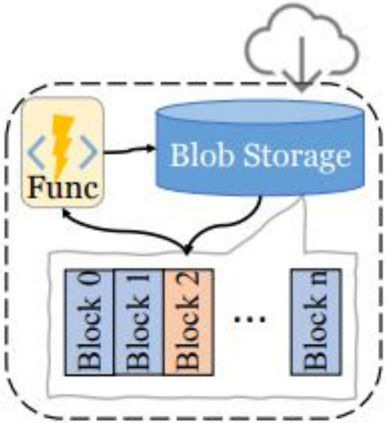


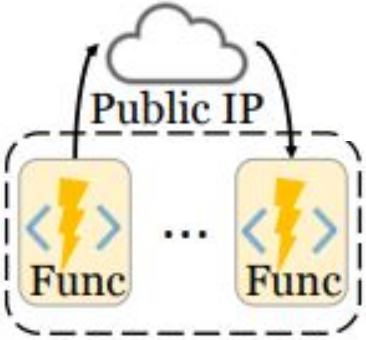
Fig. 3. Comparison of Vector Data Clustering Results using Different Unsupervised Clustering Algorithms

Naive Communication Mechanism

- Vector search requires low latency to function
- Need to communicate aggregate results between cloud functions
- Two choices:
    - Global (blob) storage
    - HTTP-based



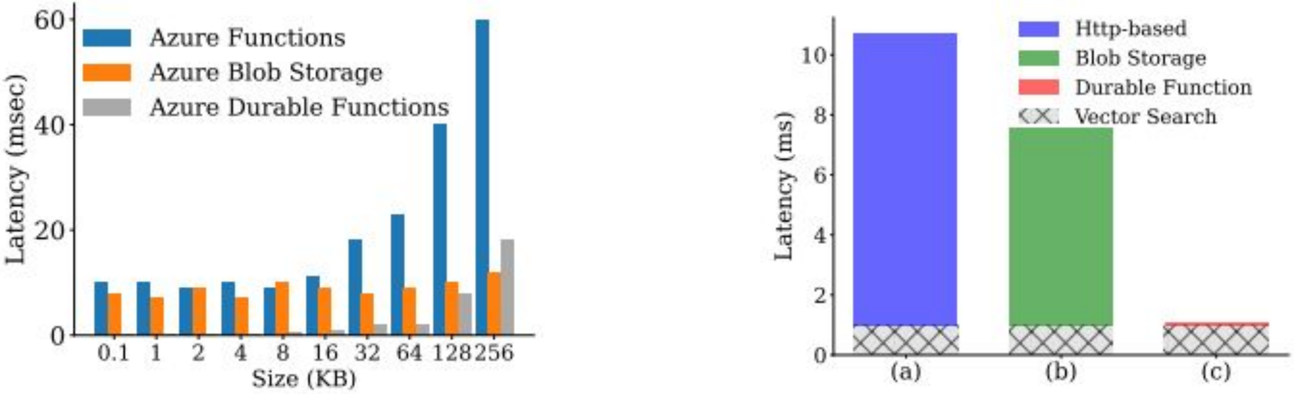
(b) Blob Storage



(a) HTTP-Based

Optimized Communication Mechanism

- Solution: Use Azure Durable functions as stateful orchestrators
- Can preserve states across multiple function invocations
- Avoid HTTP, exploit other communication channels (e.g. message passing via Azure Queue Storage

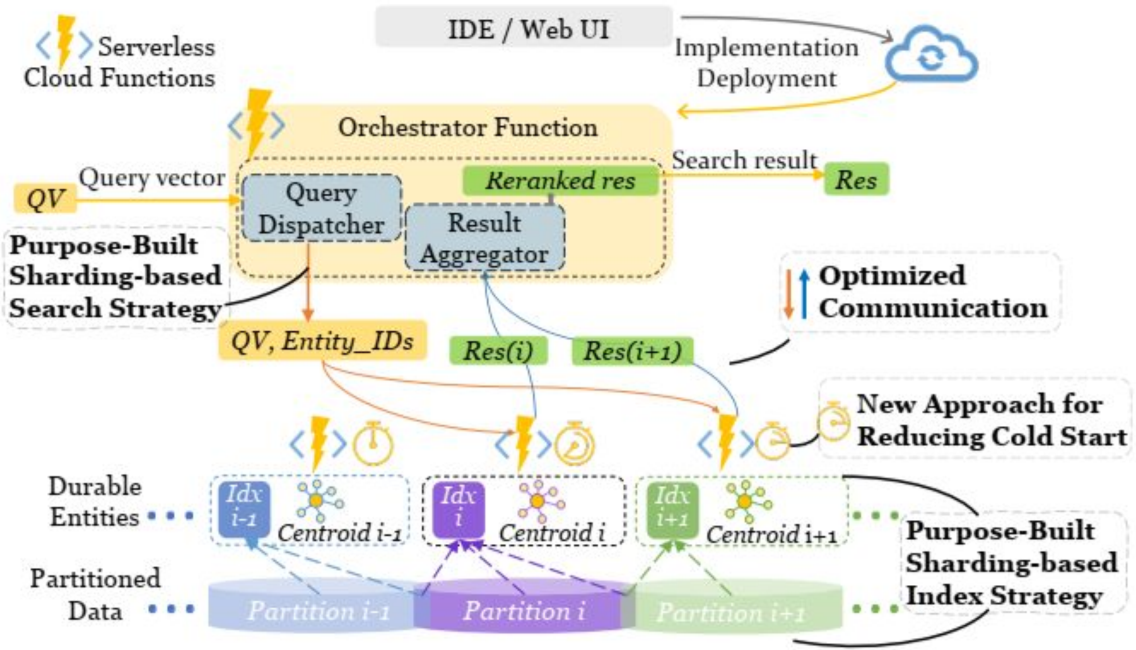


(a) Communication Overhead **(b)** Overhead with Vector Search

Fig. 4. Cloud Communication Overhead Comparison

Main System Idea

- Use a distributed vector search, treating each cloud function as a computational unit
  - Sharding
  - Communication hub



Experiment

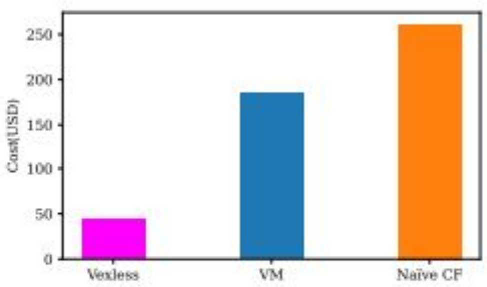
- Datasets: SIFT10M (*d=128)*, DEEP10M (*d=96*), GIST1M (*d=960*)
- Workloads: Sparse, burst, continuous, real-world
- Evaluation: Latency, accuracy (recall@10 default)
- Hardware: Azure VM F4 (8 GB RAM, 4 vCPUs)

Table 1. Datasets

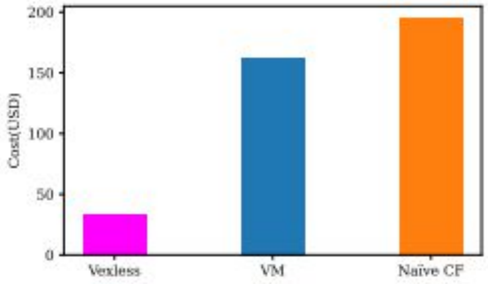
	SIFT10M	DEEP10M	GIST1M
# of Base Vectors	10,000,000	10,000,000	1,000,000
# of Query Vectors	10,000	10,000	1,000
Dimensionality	128	96	960
Data type	int32	float32	int32

Results (cost)

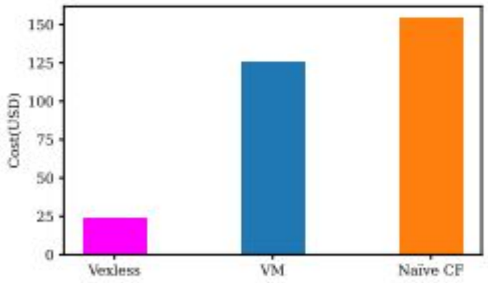
- Vexless can provide cost savings up to **5.3x** for VM-based solutions and nearly **6.5x** for naive cloud implementations



(a) SIFT10M: Monthly Cost, Sparse Queries

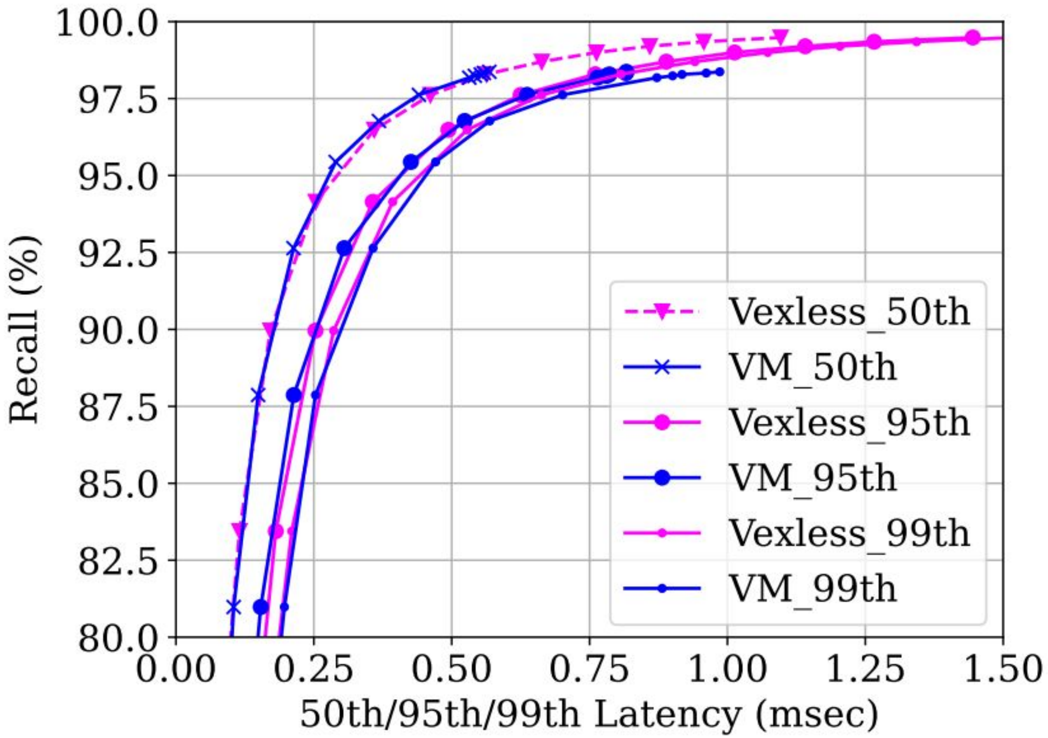


(d) DEEP10M: Monthly Cost, Sparse Queries



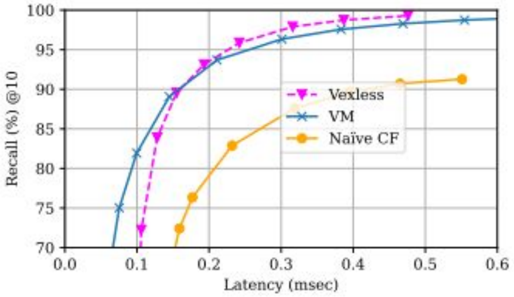
(g) GIST1M: Monthly Cost, Sparse Queries

Results (latency)

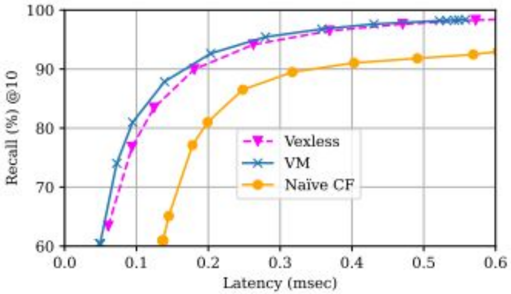


Results (recall)

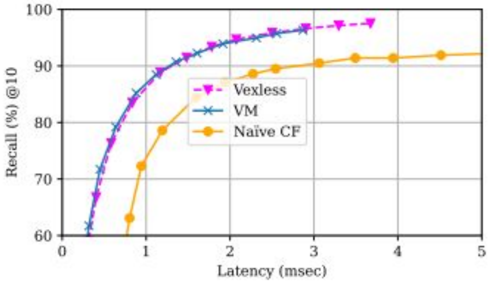
- Authors state *Vexless* offers best search performance across three different datasets



(c) SIFT10M: Search Recall@10 Performance



(f) DEEP10M: Search Recall@10 Performance



(i) GIST1M: Search Recall@10 Performance

Critical lens

After seeing all this, are you convinced *Vexless* is actually a good solution for deploying a vector database?

- Are there other ways of achieving cost savings?
- Is *Vexless* actually adhering to serverless principles?
- What are the maintenance/operational concerns?
- What about billion-point datasets necessitating multiple orchestrators?

Vexless: A Serverless Vector Data Management System Using Cloud Functions

Yongye Su, Yinqi Sun, Minjia Zhang, Jianguo Wang
Presented by Patrick Lee for CSC 2233
February 5, 2024